Mapping de tareas a cores en arquitecturas multicore. Un caso de estudio.

Fabiana Leibovich, Armando De Giusti, Marcelo Naiouf, Laura De Giusti, Franco Chichizola.

Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina.
{fleibovich, degiusti, mnaiouf, ldgiusti,francoch}@lidi.info.unlp.edu.ar

Abstract. Este trabajo analiza el mapeo manual y automático de tareas a núcleos de una arquitectura multicore, estudiando performance y overhead generado por el sistema operativo.
Se plantea un caso de estudio conocido, el problema BASIZ, que consiste en la identificación y detección de las zonas con más brillo e intensidad de color en una imagen. Este problema es de interés porque admite paralelismo funcional y de datos. Además es fácil de escalar para estudios de perfomance.
En el estudio experimental se trabaja con una arquitectura de 8 núcleos, con 2 procesadores de 4 núcleos y el problema se resuelve en forma secuencial y paralela con 2 paradigmas diferentes. En todos los casos se estudian los resultados con mapping manual y automático.
Por último, se exponen las líneas de trabajo actuales, trabajando con un cluster de multicore de 64 núcleos.

Keywords: Arquitecturas multicore. Algoritmos paralelos. Mapping. Métricas de perfomance.

1 Introducción

Reducir el tiempo de ejecución de aplicaciones con grandes requerimientos de procesamiento es el objetivo principal del procesamiento paralelo. Es decir, disminuir los tiempos de ejecución con respecto a los tiempos secuenciales.

La necesidad de reducir los tiempos de ejecución se debe al constante incremento del volumen de procesamiento y las limitaciones que impone el procesamiento secuencial en cuanto a tiempos de respuesta, acceso a datos distribuidos y manejo de la concurrencia implícita en los problemas del mundo real. [1] [2]

El procesamiento paralelo se utiliza en una gama muy amplia de aplicaciones entre las que podemos citar aplicaciones científicas, simulaciones y procesamiento de imágenes.[1]

Desde el surgimiento del primer procesador, se ha buscado incrementar la capacidad de cómputo y el rendimiento, reduciendo el tamaño físico de los procesadores. Para ello es necesario aumentar su potencia de cómputo y al mismo tiempo disminuir el tamaño de los circuitos integrados, lo que conlleva un aumento

del nivel de disipación térmica [3]. Como respuesta a esto, surgen los procesadores multicore [4].

1.1 Arquitecturas Multicore

Un procesador multicore consiste en un conjunto de núcleos de procesamiento integrados en un único chip. Normalmente, cada uno de los núcleos (core) posee su propio nivel L1 de cache y de a pares los cores comparten el nivel L2 de cache (incluido generalmente en la placa madre).

Cada procesador multicore es considerado un procesador físico, mientras que cada núcleo dentro de él es considerado un procesador lógico [5].

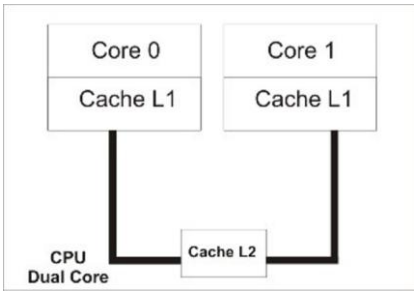

Fig. 1. Arquitectura Multicore genérica.

1.2 Caso de Estudio

La aplicación BASIZ (Bright and Saturated Image Zones) es una aplicación de procesamiento de imágenes para la identificación y detección de las zonas con más brillo e intensidad de color de una imagen.

Dada una imagen de entrada, el algoritmo lleva a cabo una serie de fases de procesamiento obteniendo como resultado la imagen original en la cual están marcadas las zonas más perceptibles al ojo humano, que son las zonas más brillantes [6][7].

Las fases de procesamiento de la aplicación están compuestas por siete pasos lógicos que pueden ser divididos en subtareas. Estos 7 pasos son [8]:

1. Separación en los tres colores básicos (rojo, verde y azul).
2. Difuminado de cada uno de los tres canales de color básico (bluring).
3. Suma de las imágenes difuminadas para cada canal de color.
4. Unión de los tres canales de color difuminados.
5. Conversión de formato de representación de la imagen difuminada de RGB a HSV (matiz, saturación y brillo).
6. Umbralización: obtener el umbral a partir del cual se determina la inclusión de los pixels a un segmento sensitivo.
7. Detección y marcado de las zonas más sensitivas al ojo humano.

Esta aplicación permite la utilización tanto de paralelismo funcional como de datos, así como también una mezcla de ambos. Esto se debe a que por un lado, la división en pasos del algoritmo permite la división de responsabilidades entre procesos.

Por otro lado, la posibilidad de tener replicado el mismo algoritmo con diferente conjunto de datos de entrada permite al mismo tiempo realizar una descomposición de datos del problema.

1.3 Estudio experimental a realizar

En esta etapa de investigación sobre la aplicación de multicores y clusters de multicores en procesamiento paralelo, nos hemos centrado en el análisis de las posibilidades de descomposición funcional y/o de datos que puede realizarse para resolver problemas (en este trabajo enfocados en el caso BASIZ) utilizando diferentes paradigmas de interacción de procesos: Master – Worker, Pipelining (descomposición funcional), así como replicación de algoritmos (descomposición de datos).

También se está realizando un análisis inicial de las diferentes maneras que existen de llevar a cabo un mapping manual de procesos a núcleos de procesamiento, teniendo en cuenta las jerarquías de memoria y la localidad de los datos.

Se trata de medir performance alcanzable con el algoritmo paralelo que resuelve BASIZ en una arquitectura multicore empleando soluciones de paralelismo funcional, paralelismo funcional y de datos y además un contraste entre el mapping que realiza el Sistema Operativo frente al mapping manual a cargo del programador. Entre las líneas actuales está la evaluación de algoritmos específicos para mapping y/o scheduling tales como AMTHA sobre este tipo de arquitectura [9].

El hardware sobre el cual se están realizando las pruebas es un Multicore Dell Poweredge 1950, que posee 2 procesadores quad core Intel Ceón e5410 de 2.33 GHz; 4 Gb de memoria RAM (compartida entre ambos procesadores); cache L2 de 6Mb entre cada par de núcleos de los procesadores. Próximamente se adquirirá un Blade de 64 cores para utilizar como cluster de multicores.

2 Contribución

La principal contribución es realizar un análisis comparativo de la performance alcanzable en una arquitectura multicore, contrastando el mapping manual y automático (Sistema Operativo) de procesos utilizando como caso de estudio un problema que se resuelve con 2 paradigmas diferentes, utilizando paralelismo funcional y de datos.

3 Estudios Experimentales Realizados

Los estudios experimentales fueron realizados en base a la implementación del algoritmo en diferentes versiones.

La solución secuencial, así como las soluciones paralelas, fueron implementadas utilizando el lenguaje C sin utilizar librerías destinadas al procesamiento de imágenes.

Las soluciones paralelas utilizan MPI como mecanismo de comunicación entre procesos, usando para ello la librería OpenMPI [10].

En las tres soluciones paralelas implementadas, se realizaron pruebas de los algoritmos con dos alternativas de mapping: en la primera, se deja a cargo del Sistema Operativo la responsabilidad de asignar procesos a núcleos de procesamiento; en la segunda realiza un mapping manual de los procesos, es decir, la responsabilidad queda a cargo del programador. En la Fig. 2 se ven los 3 esquemas de configuración adoptados para la segunda alternativa.

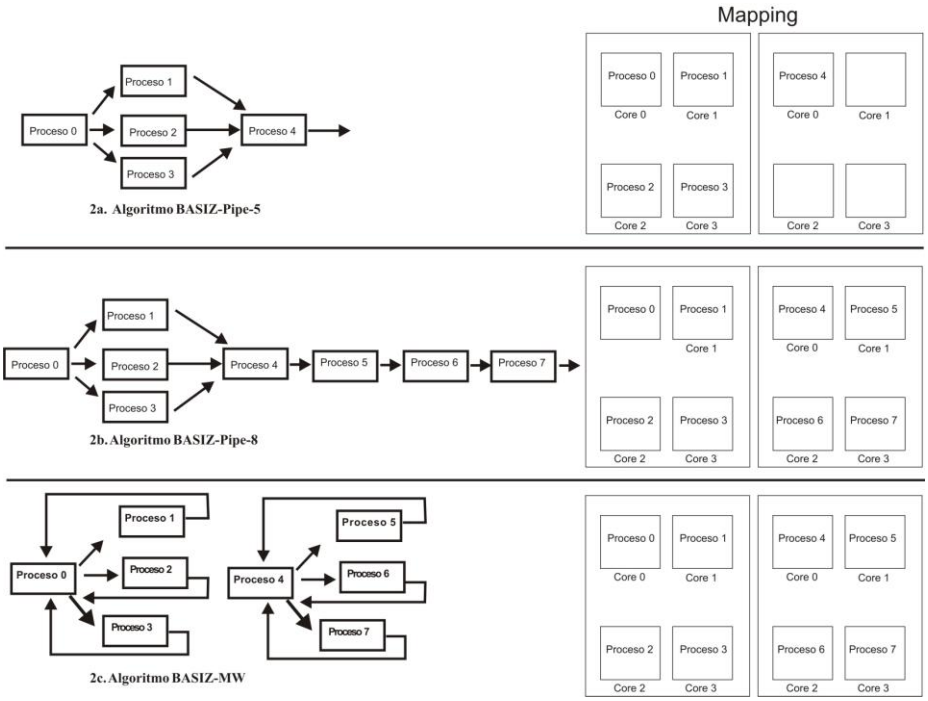

Fig. 2. Esquemas de mapping utilizados.

En la solución paralela de pipelining de 5 procesos, el mapping manual se realizó utilizando 4 núcleos de un procesador y un núcleo del segundo procesador. Por otro lado, en la solución de pipelining de 8 procesos, cada proceso fue asignado a un núcleo diferente. Figuras 2a y 2b respectivamente.

En la solución paralela Master - Worker replicada, el mapping se llevó a cabo teniendo en cuenta que se tienen 8 procesos pero que los mismos están divididos lógicamente en dos grupos de 4 procesos cada uno (replicación). Por ello el mapping se realizó asignando cada grupo de 4 procesos a un procesador diferente y dentro de cada procesador, cada proceso fue asignado a un núcleo distinto. Figura 2c.

En todos los casos, las pruebas fueron realizadas utilizando imágenes de 4096 x 4096 píxeles.

3.1 Solución Secuencial

La solución secuencial implementa el algoritmo BASIZ y ejecuta el mismo para un número de imágenes parametrizable.

BASIZ - Secuencial

```
{Para cada una de las imágenes}
1. Separar la imagen en los tres canales de color (rojo,
verde y azul) y generar una imagen para cada uno de ellos.
2. Aplicar el difuminado a los tres canales de color. Para
cada canal se aplica un filtro gaussiano de 5X5 en tres
niveles, generando una imagen para cada nivel.
3. Para cada canal de color, sumar las tres imágenes
difuminadas generando una nueva imagen.
4. Unir las tres imágenes difuminadas generando una nueva
imagen.
5. Convertir el formato de representación de la imagen recién
generada de RGB a HSV (matiz, saturación y brillo).
6. Generar el umbral a partir del cual se determinará si un
píxel pertenece o no a una zona sensitiva. Luego generar una
matriz binaria en la que cada pixel está representado por un
cero (si es un pixel no sensitivo) o un uno (si el pixel es
sensitivo).
7. En función de la matriz binaria recientemente generada,
marcar en la imagen original las zonas sensitivas.
{fin}
```

Tabla 1. Tiempos de ejecución (en segundos) obtenidos con el algoritmo secuencial para diferente cantidad de imágenes.

N° de imágenes	Tiempo de ejecución
16	1096,184349
18	1236,224733
20	1381,812570
22	1527,840057
24	1664,409866
26	1809,448193
28	1940,566717
30	2072,817969

En función de la solución secuencial, puede observarse que los pasos 2 y 3 pueden ser paralelizados para cada canal de color teniendo en cuenta que no hay dependencia de datos entre los mismos.

3.2 Solución Tipo Pipelining Usando 5 y 8 Núcleos

La primer solución tipo pipelining utiliza 5 procesos. Es una solución que utiliza paralelismo funcional.

Algoritmo BASIZ-Pipe-5

```
{Para cada una de las imágenes}
Proceso 0: Separar la imagen en los tres canales de color
(rojo, verde y azul).Para cada canal de color comunicar los
datos al proceso correspondiente.
Procesos 1, 2 y 3 ejecutan los mismos pasos cada uno para el
canal de color que le corresponde: Generar una imagen para su
canal. Aplicar un filtro gaussiano de 5X5 en tres niveles,
generando una imagen para cada nivel.
Sumar las tres imágenes difuminadas, generando una nueva
imagen. Luego comunicar estos datos al proceso 4.
Proceso 4: Unir las tres imágenes difuminadas generando una
nueva imagen. Convertir el formato de representación de la
imagen recién generada de RGB a HSV. Generar el umbral.
Generar la matriz binaria. En función de la matriz binaria
marcar en la imagen original las zonas sensitivas.
{fin}
```

En esta solución puede observarse que el procesamiento que lleva a cabo el proceso 4 puede descomponerse aún más. Por este motivo se implementó la segunda solución tipo pipelining que utiliza 8 procesos en lugar de 5.

Algoritmo BASIZ-Pipe-8

```
{Para cada una de las imágenes}
Proceso 0: Separar la imagen en los tres canales de color
(rojo, verde y azul). Para cada canal de color comunicar los
datos al proceso correspondiente.
Procesos 1, 2 y 3 ejecutan los mismos pasos cada uno para el
canal de color que le corresponde: Generar una imagen para su
canal. Aplicar un filtro gaussiano de 5X5 en tres niveles,
generando una imagen para cada nivel. Sumar las tres imágenes
difuminadas, generando una nueva imagen. Luego comunicar
estos datos al proceso 4.
Proceso 4: Unir las tres imágenes difuminadas generando una
nueva imagen. Comunicar los datos de cada canal al proceso 5.
```

```
Proceso 5: Convertir el formato de representación de la
imagen recién generada de RGB a HSV. Comunicar la matriz de
brillo al proceso 6.
Proceso 6: Generar el umbral. Generar la matriz binaria.
Comunicar la matriz binaria al proceso 7.
Proceso 7: En función de la matriz binaria marcar en la
imagen original las zonas sensitivas.
{fin}
```

Tabla 2. Tiempos de ejecución (en segundos) obtenidos con los algoritmos *BASIZ-Pipe-5* y *BASIZ-Pipe-8*

N° de imágenes	Tiempo de ejecución mapping Sistema Operativo BASIZ-Pipe-5	Tiempo de ejecución mapping manual BASIZ-Pipe-5	Tiempo de ejecución mapping Sistema Operativo BASIZ-Pipe-8	Tiempo de ejecución mapping manual BASIZ-Pipe-8
16	347,115726	352,381197	350,404372	350,953746
18	396,351962	391,463422	390,577530	392,367580
20	432,877443	434,128574	433,501823	435,226090
22	479,019390	486,018654	475,780649	484,694018
24	520,146222	518,409603	524,983332	521,461347
26	561,853795	562,451366	564,203506	563,111458
28	602,108146	606,043863	610,901599	608,587703
30	645,601332	647,988962	647,120775	649,378497

3.3 Solución Master-Worker Replicada

Esta solución está implementada utilizando el paradigma de interacción de procesos Master – Worker. Lleva a cabo una mezcla de paralelismo funcional y paralelismo de datos (replicación), replicando una vez el siguiente algoritmo:

Algoritmo BASIZ-MW

```
{Para cada una de las imágenes}
Proceso 0: Separar la imagen en los tres canales de color
(rojo, verde y azul). Para cada canal de color comunicar los
datos al proceso correspondiente.
Recibir los datos correspondientes a la unión de las imágenes
difuminadas para cada canal de color. Unir las tres imágenes
difuminadas generando una nueva imagen. Convertir el formato
de representación de la imagen recién generada de RGB a HSV.
Generar el umbral. Generar la matriz binaria.
En función de la matriz binaria marcar en la imagen original
las zonas sensitivas.
Procesos 1, 2 y 3 ejecutan los mismos pasos cada uno para el
canal de color que le corresponde: Generar una imagen para su
```

```
canal. Aplicar un filtro gaussiano de 5X5 en tres niveles,
generando una imagen para cada nivel. Sumar las tres imágenes
difuminadas, generando una nueva imagen. Luego comunicar
estos datos al proceso 0.
{fin}
```

Tabla 3. Tiempos de ejecución (en segundos) obtenidos con el algoritmo BASIZ-MW para diferente cantidad de imágenes.

N° de imágenes	Tiempo de ejecución mapping Sistema Operativo	Tiempo de ejecución mapping manual
16	252,665408	249,079594
18	281,888035	280,753140
20	314,353628	316,070032
22	350,136947	342,358989
24	378,151115	376,240835
26	414,803686	409,411892
28	437,716598	437,219655
30	473,109013	462,835451

4 Resultados comparados

Como resultado de las experimentaciones realizadas, se observa a continuación un gráfico del speedUp y la eficiencia de los algoritmos recientemente explicados. En él se contrastan el speedUp y la eficiencia obtenidos con mapping automático (Sistema Operativo) y con mapping manual.

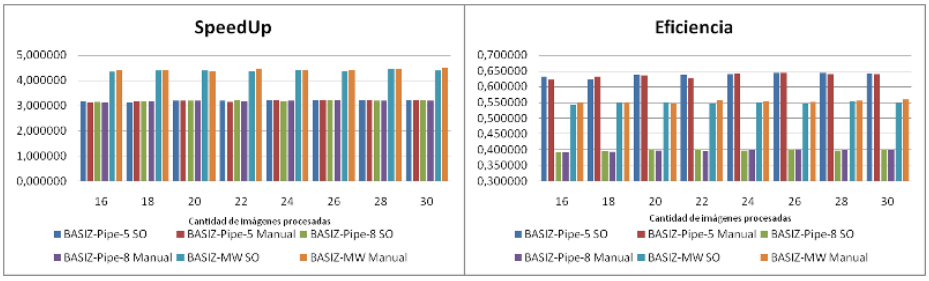

Fig. 3. Comparación de speedUp y eficienciaentre mapping automático y manual.

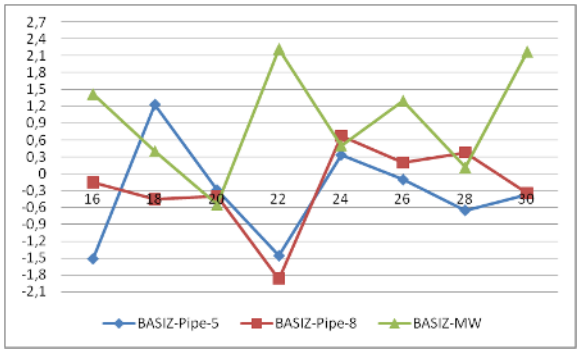

Fig. 4. Porcentaje de diferencia entre la asignación automática y manual de las tres alternativas: ((Tiempo automático – Tiempo manual) *100/Tiempo automático).

5 Análisis de los resultados obtenidos

- Debido a que los tiempos de ejecución dependen del tamaño de la imagen que se utilice y no del contenido de la misma, el problema es escalable incrementando el número de imágenes.
- Los resultados obtenidos muestran que el speedup del *Algoritmo BASIZ-Pipe-8* no mejora con respecto al speedup del *Algoritmo BASIZ-Pipe-5*. Esto se debe a que la descomposición funcional realizada está desbalanceada. Esto provoca que la fase más extensa en tiempo de ejecución (filtro gaussiano) no permita a los procesos que le siguen en el pipe trabajar de manera continua explotando el paralelismo, ya que como el tiempo de ejecución de los mismos es menor, deben pasar parte del tiempo esperando a que la fase más extensa termine para poder continuar procesando.
- Por último, los resultados muestran que en el *Algoritmo BASIZ-MW* el speedup es mayor que en las demás soluciones dada la replicación que permite procesar más de una imagen simultáneamente. Además, puede observarse que el mapping manual de procesos mejora el speedup si se compara con el mapping realizado por el Sistema Operativo. Esto se debe a que la replicación, en la que lógicamente existen dos grupos de 4 procesos cada uno, permite mapear cada grupo a un procesador distinto de manera que los procesos de un mismo grupo comparten el mismo procesador físico y la misma caché.

6 Conclusiones y Líneas de trabajo futuras

Se ha realizado un estudio experimental comparativo de la performance alcanzable en una arquitectura multicore, contrastando el mapping manual y automático (Sistema Operativo) de procesos utilizando como caso de estudio el problema BASIZ resuelto de diferentes maneras, utilizando paralelismo funcional y de datos.

La principal línea de investigación actual es la extensión del trabajo experimental a un cluster de multicores (que pueden ser heterogéneos) y la comparación de diferentes algoritmos de mapping y scheduling orientados a tareas paralelas, tales como AMTHA.

7 Referencias

1. Grama A., Gupta A., Karypis G., Kumar V., "An Introduction to Parallel Computing. Design and Analysis of Algorithms. 2nd Edition". Pearson Addison Wesley (2003).
2. Leopold C., "Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches". Wiley, New York (2001).
3. Guccione S. "Multicore Devices: A New Generation of Reconfigurable Architectures".
4. AMD. "Evolución de la tecnología de múltiple núcleo". http://multicore.amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution.aspx (2009).
5. Burger T. "Intel Multi-Core Processors: Quick Reference Guide "http://cachewww.intel.com/cd/00/00/23/19/231912_231912.pdf.
6. Roig C., Ripoll A., Borrás J., Luque E. "Efficient Mapping for Message-Passing.Applications Using the TTIG Model: A Case Study in Image Processing."
7. Roig C., Ripoll A., Senar M. A., Guirado F., Luque E. "A New Model for Static Mapping of Parallel Applications with Task and Data Parallelism".
8. Roig C. "Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos" Tesis Doctoral – 2005 Unidad Autónoma de Barcelona –España.
9. De Giusti L. "Mapping sobre Arquitecturas Heterogéneas". Tesis Doctoral, Universidad Nacional de La Plata (2008).
10. http://www.open-mpi.org